2013 IEEE Intelligent Vehicles Symposium (IV)
June 23-26, 2013, Gold Coast, Australia

Optimal Longitudinal Control Planning with Moving Obstacles

Jeff Johnson and Kris Hauser

***Abstract*— At intersections and in merging traffic, intelligent road vehicles must solve challenging optimal control problems in real-time to navigate reliably around moving obstacles. We present a complete planner that computes collision-free, optimal longitudinal control sequences (acceleration and braking) using a novel visibility graph approach that analytically computes the reachable subset of path-velocity-time space. We demonstrate that our method plans over an order of magnitude faster than previous approaches, making it scalable and fast enough (tenths of a second on a PC) to be called repeatedly on-line. We demonstrate applications to autonomous driving and vehicle collision warning systems with many moving obstacles.**

I. INTRODUCTION

The Federal Highway Administration in the United States notes that the frequency of automobile collisions is directly related to the number of *conflict points* [1], which are points in a vehicle's path that are crossed by the paths of obstacles, such as pedestrians, bicyclists, and other vehicles. Complex urban intersections may involve dozens of conflict points and require an intelligent vehicle to simultaneously avoid leading/merging vehicles, cross-traffic, and pedestrians. While specialized collision avoidance techniques have addressed specific conflict types, such as following behavior [2], merging [3], and cross-traffic [4], there is little work on techniques that address heterogeneous conflict types.

To address this problem we present a planner that extends an analytical approach previously developed for negotiating cross-traffic [4]. Given a desired vehicle path and estimates of future obstacle behavior, our planner computes a *visibility graph* that represents the set of all possible path/velocity/time (PVT) states that are reachable via a collision-free longitudinal control sequence. The optimal control sequence is extracted directly from this graph (Fig. 1).

We present two technical contributions beyond our prior work. The first is an improved method for computing the *PT obstacles*, which represent the set of colliding states in the path-time plane. Our prior paper approximated PT obstacles as axis-aligned rectangles, which works well for cross-traffic but is a poor approximation for leading and merging vehicles. Instead, the new approach computes an arbitrarily close polygonal approximation to the true PT obstacle, and the approximation can be tuned to trade off between speed and accuracy. The second is an extension allowing the planner to handle the diagonal edges that emerge from our new PT obstacle construction. This substantial extension allows our method to incorporate vehicle following into planning.

The planner now handles arbitrary vehicle paths, polygonal vehicle and obstacle models, and arbitrary velocity profiles. It also naturally handles uncertainty by "growing" the obstacles according to confidence bounds on future behavior. At over an order of magnitude faster than competing approaches, it can handle many moving obstacles in tenths of a second on a standard PC. We demonstrate its application to a simulated urban intersection involving pedestrians, bicyclists, and automobiles, as well as merging with cross-traffic on a rural highway.

Jeff Johnson and Kris Hauser are with the School of Informatics and Computing, Indiana University {jj56,hauserk}@indiana.edu

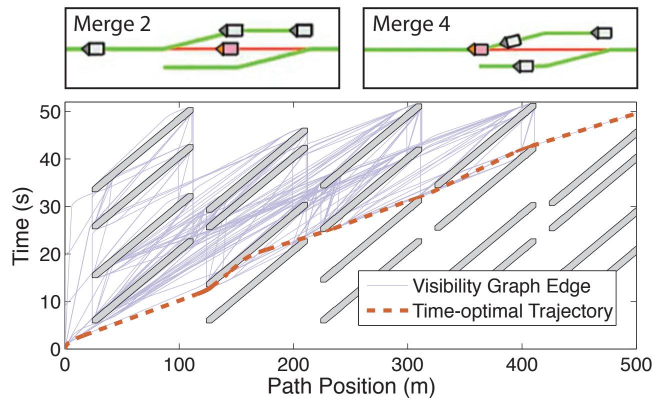


Fig. 1: Simulation of 20 vehicles merging onto/off of the driver's path. Above: the driver (red) passes two of the merges. Below: Diagonal PT obstacles from the shared obstacle/driver paths. The supplemental video contains a simulation of this scenario.

II. RELATED WORK

Navigation among moving obstacles is a challenging problem with a long history. In one class of problems, the vehicle has choice over spatial and velocity controls. Unfortunately, this problem is intractable: in fact, even among static obstacles the general 2D motion planning problem is PSPACE-hard [5]. In time-varying domains, a popular approach is to plan conservatively using velocity obstacles [6], [7], which are sets of velocities that will lead to collision when obstacle velocities are maintained. More recently, randomized techniques have been used. Kindel, et. al. [8] employed a variant of Probabilistic Roadmaps [9] with fast replanning to plan arbitrary trajectories in the presence moving obstacles and sensor uncertainty. The MIT team [10] for the 2007 DARPA Urban Challenge used a real-time implementation of Rapidly-exploring Random Trees [11] to plan vehicle trajectories. Randomized methods achieve tractability but sacrifice hard guarantees on completeness and optimality, which can be problematic for road vehicles that must operate near 100% reliably even in heavily crowded environments.

978-1-4673-2754-1/13/$31.00 ©2013 IEEE

A second class of problems assumes a structured road network with only a handful of paths available, and that the vehicle may track a given path with reasonably high accuracy [12], [13], [14]. These *longitudinal control* problems achieve tractability by decoupling spatial path planning and velocity control. The Carnegie-Mellon team in the 2007 DARPA Urban Challenge exploited path/velocity decomposition for road lane navigation by computing optimal paths based on the centerline of the road lanes [15]. Their method then proposes candidate trajectories along the optimal path and chooses the best according to various metrics, including how well they avoid static and dynamic obstacles. Like other sample-based methods, it cannot guarantee a solution.

Exact methods for the longitudinal control planning problem include the *visibility graph* [16] and explicit search over a discretized state space [17]. Our method is a visibility graph that addresses the major shortcoming of [16] by incorporating acceleration bounds. Unlike explicit discretization [17], our method is analytical and runs in polynomial time, making it fast enough to be used for real-time replanning. In addition, the visibility graph construction allows the exact set of velocity obstacles to be efficiently computed.

III. PROBLEM DEFINITION

The vehicle is assumed to travel along a known path and to sense objects in its environment and estimate their intended behavior over a fixed time horizon, e.g., using vision, radar, or inter-vehicle communication. The planner is then asked to solve a longitudinal control problem to define the vehicle's future trajectory over this horizon (in the case of an intelligent vehicle) or to deliver a collision warning (for a driver assistance system). The planner is designed to be invoked repeatedly in a model predictive control-like scheme to advance the horizon and respond to changing sensor input. We define the planner's assumptions below.

Inputs. Our planner takes as input the robot R's arc-length parameterized path $P_R(p) : [0, p_{max}] \mapsto \mathcal{C}$, and a list of n obstacles O_i along with their predicted paths $P_{O_i}(p)$, $i = 1, \ldots, n$. All vehicles are modeled as polygons that translate and rotate as they slide along piecewise linear paths. We assume the orientation of a vehicle at any position p along its path is always tangent to the path, and hence its world-space layout is entirely determined by p. Uncertain obstacle behavior is handled by specifying an interval of path positions $p_i(t) \in [\underline{p}_i(t), \overline{p}_i(t)]$ at which obstacle i might lie at time t (Figs. 2e & 2f). We also make the simplifying assumption that obstacle behavior is independent of driver behavior.

Dynamically feasible PVT trajectories. The planner computes a continuous curve in the *path-velocity-time* (PVT) state space, in which states $x = (p, v, t)$ consist of a path position $p \in [0, p_{max}]$, velocity v, and time $t \in [0, t_{max}]$. Dynamic constraints include velocity and acceleration bounds:

$$\begin{aligned} &\dot{p} = v \\ &v \in [\underline{v}, \overline{v}] \;:\; \underline{v} \geq 0 \\ &\dot{v} \in [\underline{a}, \overline{a}] \;:\; \underline{a} < 0 < \overline{a} \end{aligned}$$

A trajectory $x(t) = (p(t), v(t), t)$ defined over $t \in [a, b]$ that satisfies these conditions for all $t \in [a, b]$ is called *dynamically feasible*.

PT obstacles. Each obstacle O_i imposes a forbidden region CO_i in the PT plane that corresponds to the (p, t) points that would cause the driver and obstacle geometry to overlap [17]. In other words, $CO_i = \{(p, t) \mid (T_R(p)R \cap T_{O_i}(p_i(t))O_i) \neq \emptyset\}$ where $T_R(p)$ (resp., $T_{O_i}(p)$) are the transformations that translate and rotate a vehicle's polygon to the driver's (resp., obstacle's) path at position p. With uncertain obstacle behavior we take PT obstacles to be the union of obstacles over all possible path positions $p_i(t) \in [\underline{p}_i(t), \overline{p}_i(t)]$. A trajectory $x(t)$ is *collision free* if $(p(t), t) \notin CO_i$ for all $i = 1, \ldots, n$ and $t \in [a, b]$, and it is *feasible* if it is both dynamically feasible and collision free.

Boundary conditions. The planner must generate a path from the initial state $x(0) = (0, v_0, 0)$ where v_0 is the vehicle's current velocity to one of two goal cases:

1) Successful navigation: $p(t_{end}) = p_{max}$ and $v(t_{end}) \in [\underline{v}_{goal}, \overline{v}_{goal}]$ for some $t_{end} \leq t_{max}$. We allow a range of goal velocities to obey bounds on speed limits.
2) Premature stop: $p(t_{max}) < p_{max}$ and $v(t_{max}) = 0$. This case can occur when lead vehicles are stopped.

The planner will output the Case 1 solution of minimum time if one exists, or the Case 2 solution of furthest progress.

IV. PLANNING SYSTEM

The planner consists of two parts. First, it computes an approximation to the PT obstacles. Then, it builds a visibility graph in PVT space and constructs the optimal trajectory by searching the graph.

A. PT Obstacle Construction

It is challenging to compute the boundaries of PT obstacles exactly because they may be arbitrarily curved. A simple approximation technique might build a grid in PT space with resolution τ and test each cell for collision, but this requires discretization in two dimensions and a cost of $O((t_{max} p_{max})/\tau^2)$. Instead, our approach discretizes only the time dimension and *analytically* computes forbidden intervals in the path dimension. With a computational cost of $O(t_{max}/\tau)$ this technique leads to significant savings, and the approximation approaches the true PT obstacle as $\tau \to 0$.

We present the computation for a single obstacle O. Consider a uniform grid on the time dimension $0, \tau, 2\tau, \ldots, t_{max}$. Our algorithm computes the left and rightmost extent of CO_i within each horizontal strip $t \in [k\tau, (k+1)\tau]$ in the PT plane, resulting in a conservative *forbidden rectangle* $[a_k, b_k] \times [k\tau, (k+1)\tau]$. Fig. 2a shows PT obstacles constructed at progressively finer resolutions. The rectangles are then wrapped with a polygon to smooth their jagged edges as shown in Fig. 2b. This procedure is listed in Algorithm 1 and relies on two key subroutines.

Subroutines. The *SweptVolume* subroutine computes a conservative approximation to the world space S_W swept out by the obstacle between times $k\tau$ and $(k+1)\tau$. Let $\underline{W}$ and $\overline{W}$ be the models of O at the minimum and

Algorithm 1 BuildPTObstacle$(O, P_O, \tau, t_{max}, \underline{p}_O, \overline{p}_O)$

1: $CO \leftarrow \emptyset$
2: **for** $t = 0, \tau, 2\tau, \ldots, t_{max} - \tau$ **do**
3: $\quad S_W \leftarrow SweptVolume(O, P_O, \underline{p}_O(t), \overline{p}_O(t+\tau))$
4: $\quad [a,b] \leftarrow ComputeForbiddenInterval(S_W, R, P_R)$
5: $\quad CO \leftarrow CO \cup [a,b] \times [t, t+\tau]$
6: **end for**
7: **return** $BoundingPolygon(CO)$

maximum possible path extents at times $k\tau$ and $(k+1)\tau$ (i.e., $\underline{W} = T_O(\underline{p}_O(k\tau))O$ and $\overline{W} = T_O(\overline{p}_O((k+1)\tau))O$ assuming obstacles move monotonically along their paths). When $\underline{W}$ and $\overline{W}$ are on the same segment of the path P_O, $S_W = Conv(\underline{W}, \overline{W})$ where $Conv$ denotes the convex hull (Fig. 2c). When they lie on adjacent segments, additional models W_{θ_1} and W_{θ_2} are computed at the path segment junction, oriented at angles θ_1 of the previous segment and θ_2 of the next. We must also account for the intermediate postures of the rotation, because each vertex of O sweeps out an arc centered at the junction. Tangents of the arc endpoints are computed for each vertex, and the intersection point of the tangents is added to a set of points $\mathcal{S}$. We then let $W_\theta = Conv(W_{\theta_1}, \mathcal{S}, W_{\theta_2})$, as in Fig. 2d. S_W is then the union of of $Conv(\underline{W}, W_{\theta_1})$, W_θ, and $Conv(W_{\theta_2}, \overline{W})$. This construction generalizes to multiple traversed path segments in a straightforward manner.

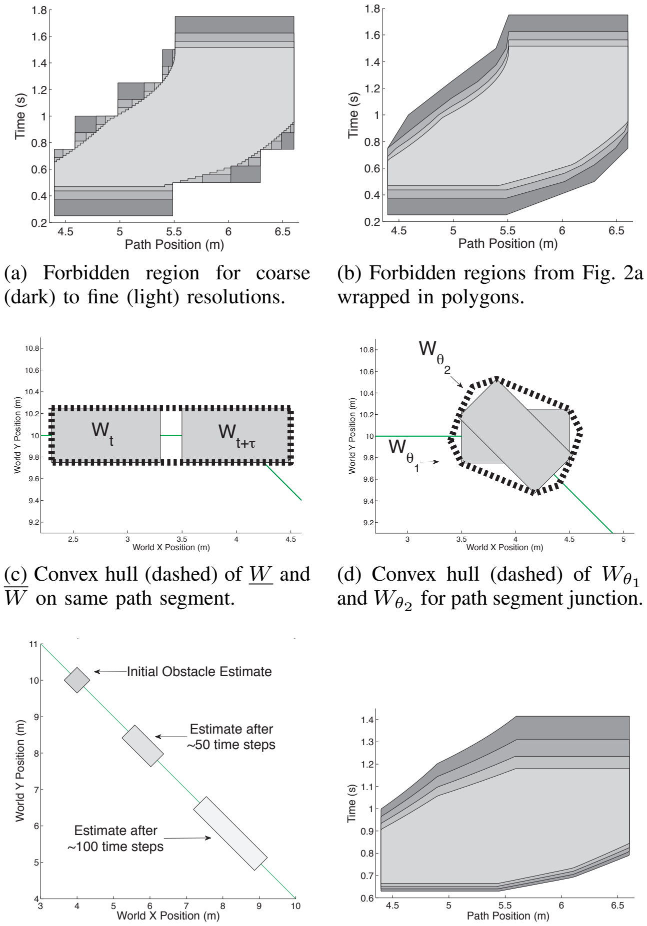
The *ComputeForbiddenInterval* subroutine computes the points a_k and b_k at which the driver R first comes into contact with a swept world space obstacle S_W, and last leaves contact with S_W. At each extremum, it is either the case that a vertex of R lies on an edge of S_W, or that a vertex of S_W lies on an edge of R. A search of all vertex-edge combinations will then find all such extrema along each line segment of P_R, and a_k and b_k are output as the minimum and maximum of these extrema.

The last step of the procedure (Line 7) wraps the forbidden rectangles with a polygon. This can dramatically reduce the number of vertices in the PT obstacles, yielding significantly faster visibility graph construction. During this procedure, we discard all interior vertices and those vertices whose incoming and outgoing edges are of the same slope.

Complexity analysis. Let $k = t_{max}/\tau$ be the number of grid points. Assuming $|R|$ and $|O|$ are bounded by m, $SweptVolume$ has average case running time $O(m \log m |P_O|/k)$, $ComputeForbiddenInterval$ is $O(|P_R| m^2)$, and $BoundingPolygon$ is $O(k)$. Overall, complexity is $O(|P_R| m^2 k + |P_O| m \log m)$.

B. Visibility Graph Planner

The planner now proceeds from the observation that any time-optimal trajectory will either connect trivially to the goal, or be tangential to a forbidden region on the PT plane. Because there will always be a time-optimal trajectory if the path is traversable, searching among tangential trajectories is guaranteed to find a solution if one exists. To search



(a) Forbidden region for coarse (dark) to fine (light) resolutions.

(b) Forbidden regions from Fig. 2a wrapped in polygons.

(c) Convex hull (dashed) of $\underline{W}$ and $\overline{W}$ on same path segment.

(d) Convex hull (dashed) of W_{θ_1} and W_{θ_2} for path segment junction.

(e) Obstacle estimate growing over time due to uncertainty.

(f) PT obstacle estimates for large (dark) to small (light) uncertainty.

Fig. 2: Illustrating PT obstacle construction.

tangential trajectories, the planner computes sets of reachable velocities at each PT obstacle vertex by constructing a visibility graph. Once all the reachable velocity sets are computed, they can be traversed backwards towards the origin to construct a feasible trajectory.

1) Reachable Velocity Sets within Homotopy Channels: A *homotopy channel* H is a region in PT space given by upper and lower bounds $l(t) \leq p(t) \leq u(t)$. The mathematical principle behind our approach is that all extremizing trajectories from one PVT point to another through a channel are a combination of bang-bang motions in free-space and portions that are tangent to $l(t)$ and $u(t)$ [18]. In [4] we proved that the set of reachable velocities from a starting state $x_1 = (p_1, v_1, t_1)$ through a single H and reaching a goal PT point (p_g, t_g) is convex. Hence it suffices to find the velocity-extremizing trajectories in H ending at (p_g, t_g).

In our prior work with rectangular PT obstacles we demonstrated that it was sufficient to incrementally propagate the extreme velocities of free-space trajectories from x_1 to (p_g, t_g) and each intermediate vertex $x_2, x_3, \ldots, x_k$ in H, then propagate from x_2 to (p_g, t_g) and $x_3, \ldots, x_k$, and so on through. We do not need to compute motions tangent

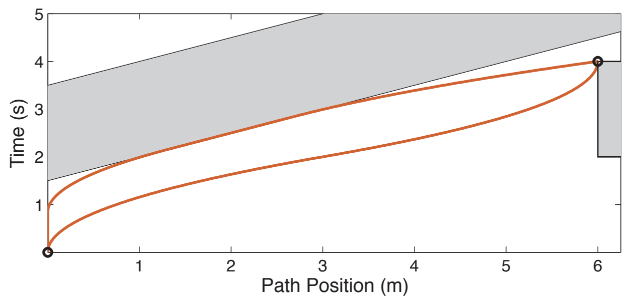


Fig. 3: Upper and lower feasible trajectories (curves) to the upper right point, with $v_0 \in [0, 0]$ and $\dot{v} \in [-1.5, 1.5]$.

to obstacles because such motions would necessarily pass through a vertex on the boundary of H (or (p_g, t_g) itself, if it were to lie on the boundary).

With PT obstacles that are arbitrary polygons, however, we must not ignore the possibility of optimal motions that pass tangent to obstacles (Fig. 3). We will now describe how we extend velocity propagation to consider diagonal edges.

2) Velocity Propagation amongst Diagonal Edges: This section presents the basic Velocity Interval Propagation (VIP) subroutine in our planner. Let (p_1, t_1) and (p_2, t_2) be two points in the PT plane (these are typically obstacle vertices), and V_1 be an interval of initial velocities. The output of the routine is the interval of velocities V_2 attainable at (p_2, t_2) by feasible trajectories starting at (p_1, v_1, t_1), where v_1 ranges over all V_1. We will describe how to compute the maximum of V_2; the minimum is symmetric.

It is a straightforward matter of algebra to compute velocity-maximizing solution $U(t)$ in the obstacle-free case analytically; it is a combination of parabolic and linear segments (see [4] for details). We then perform collision checking on $U(t)$. There are three cases to handle:

1) If it is collision-free, then it is optimal and VIP outputs $\dot{U}(t_2)$ as the maximum velocity.
2) If it collides with a vertical, horizontal, or negatively sloping PT obstacle edge, VIP outputs nothing.
3) If it collides with a diagonal edge with positive slope, VIP performs further processing, described below.

Case 1 is obviously correct. In Case 2 it has been shown that either there is no feasible trajectory or there is a velocity-maximizing trajectory that passes through one of the endpoints of that PT obstacle edge [4]. In this latter case, the planner will find the correct trajectory by propagating from (p_1, t_1) to that vertex and then from that vertex to (p_2, t_2), and hence it is safe to discard this particular propagation.

The remaining Case 3 requires further examination because it may be possible for an optimal trajectory to follow this diagonal edge tangentially. Hence we need to consider generating optimal trajectories from points to edges and back to points. Moreover, it may be possible (although rare) for the trajectory to touch *several* diagonal edges, and hence we must consider any number of edge-to-edge trajectories.

To do so we use a recursive procedure that makes use of the following subroutines, which construct optimal trajectories between various primitives in the absence of obstacles:

1) $PointToEdge(p_1, t_1, V_1, e)$: accepts an initial point (p_1, t_1) with velocity interval V_1 and edge e, and builds a time-optimal trajectory $(p_1, t_1) \rightarrow e$ that terminates at a point of tangency to e without crossing.
2) $EdgeToPoint(e, p_e, p_2, t_2)$: accepts an initial point p_e along edge e and builds a trajectory $e \rightarrow (p_2, t_2)$ that maximizes arrival velocity without crossing e.
3) $EdgeToEdge(p_{e_1}, e_1, e_2)$: accepts an initial point p_{e_1} along edge e_1 and builds a trajectory $p_1 \rightarrow e_2$ that terminates at a point of tangency to e_2 without crossing e_1 or e_2. The trajectory is constructed such that the time before e_1 is departed is minimized.

The algorithm is given below:.

Algorithm 2 MaximumTerminalVelocity_Diagonal(e)

$T_1 \leftarrow MTV\_P2E(p_1, t_1, V_1, e)$
if $T_1 = \emptyset$, **then return** NIL
$T_2 \leftarrow MTV\_E2P(e, FinalPoint(T_1), p_2, t_2)$
if $T_2 = \emptyset$, **then return** NIL
return $\dot{T}_2(t_2)$

It calls two subroutines, MTV_P2E that recursively computes a time-optimal, feasible trajectory from (p_1, t_1) to the diagonal edge e, and MTV_E2P that recursively computes a velocity-maximizing, feasible trajectory from e to (p_2, t_2). Both may call a subroutine MTV_E2E that recursively computes a time-optimal, feasible trajectory between two edges e and e'. Pseudocode for MTV_P2E is provided below. MTV_E2E and MTV_E2P are very similar.

Algorithm 3 MTV_P2E(p_1, t_1, V_1, e): Recursively compute a time-optimal trajectory from p_1 to edge e given initial velocity range V_1.

$T_1 \leftarrow PointToEdge(p_1, t_1, V_1, e)$
if $T_1 = \emptyset$, **then return** $\emptyset$
$e' \leftarrow InitialCollidingEdge(T_1)$
if $e' = NIL$, **then return** T_1
$T_1 \leftarrow MTV\_P2E(p_1, t_1 e', V_1)$
if $T_1 = \emptyset$, **then return** $\emptyset$
$T_2 \leftarrow MTV\_E2E(FinalPoint(T_1), e, e')$
if $T_2 = \emptyset$, **then return** $\emptyset$
return $T_1 \rightarrow T_2$

Fig. 3 shows an instance of the output of VIP with a diagonal edge defining the maximum terminal velocity.

3) Visibility Graph Construction: Now we present the method for computing reachable velocities outside of a PT channel. Although the reachable velocities at a vertex form a convex set within a single channel, for multiple channels this set may in fact be disjoint. Note that for n obstacles there are in general 2^n possible channels, which raises the possibility that the problem is exponentially hard. However, we showed in [4] that the number of disjoint velocity intervals any PT point is effectively bounded by a small constant, and hence a visibility graph can be computed in polynomial time.

The visibility graph vertices consist of a vertex representing the start state, PT obstacle vertices, and a vertex representing terminal states. Edges store representative trajectories that define the min and max velocity trajectories between two vertices. Planning proceeds incrementally by calling VIP between all pairs of vertices, in sorted order of in increasing p. Let these vertices be $(p_0, t_0), (p_1, t_1), \ldots, (p_N, t_N)$ with $(p_0, t_0) = (0, 0)$ the initial state, and let $V_0 = [v_0, v_0]$. Each vertex (p_j, t_j) stores a set of reachable velocities V_j, represented as a list of zero or more disjoint intervals. The algorithm is given below:

Algorithm 4 BuildVG: computes the reachable velocities V_i at each point (p_i, t_i) and at the goal.

for $j = 1, \ldots, N$ **do**
  for $i = 0, \ldots, j-1$ **do**
    for each disjoint interval $[a, b]$ in V_i **do**
      Call *Propagate*$([a, b], i, j)$
    end for
  end for
  $V_j \leftarrow$ *Merge*(S_j)
  for each disjoint interval $[a, b]$ in V_j **do**
    Call *PropagateGoal*$([a, b], j)$
  end for
end for

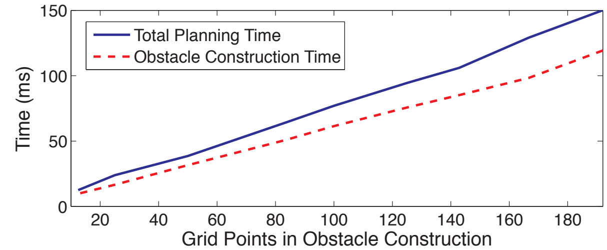
The *Merge* step is crucial to maintaining polynomial-time complexity; without it, planning would be exponential-time. During a merge, care must be taken to maintain the homotopy classes of each convex interval in each V_j; it is safe to take the convex hull of intervals arriving at (p_j, t_j) within the same homotopy suffix due to the convexity property, but not otherwise. These implementation details can be found in [4].

Once the visibility graph is computed, a time-optimal trajectory is recovered by tracking backwards from the goal vertex. (Note that the graph is a complete representation of *all* feasible trajectories, so it may also be possible to optimize other objective functions like maximum safety.)

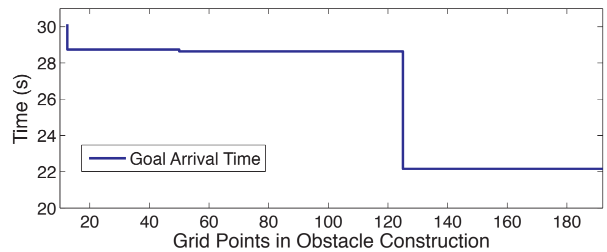
4) Complexity Analysis: Complexity is bounded by BuildVG and depends mainly on the number of PT obstacle vertices N. BuildVG makes $O(cN^2)$ calls to *Propagate* where c is the average number of disjoint velocity intervals at a vertex. For all practical purposes, c is a small constant, but in highly pathological cases it can be $O(N)$. *Propagate* constructs a candidate trajectory in $O(1)$ time and performs an $O(N)$ collision detection. If a diagonal edge is hit, two or more additional trajectory generation and collision detection routines are called. In all, *Propagate* is $O(dN)$ where d is the number of diagonal edges hit during the construction. In practice d is a small constant, but in the worst case it may be $O(N)$. The resulting complexity is $O(cdN^3)$.

C. Empirical Performance

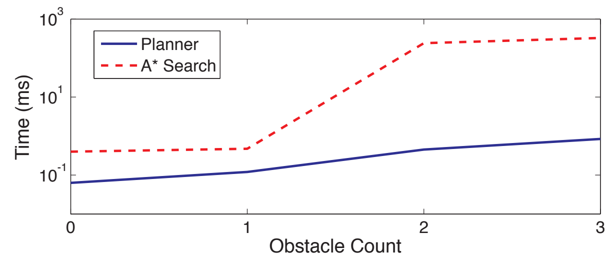
Fig. 5 shows empirical performance of PT obstacle construction and visibility graph construction for a scenario similar to that in Fig. 4. Results are averages of 10 runs on a single core of a 2.3GHz PC. In Fig. 5a the planner is run with



(a) Running times for obstacle construction and planning for increasingly fine obstacle discretization



(b) Trade-off between obstacle discretization and optimality of plan.



(c) Comparison against A^* search in toy scenario. (Note the logarithmic scale on the time axis).

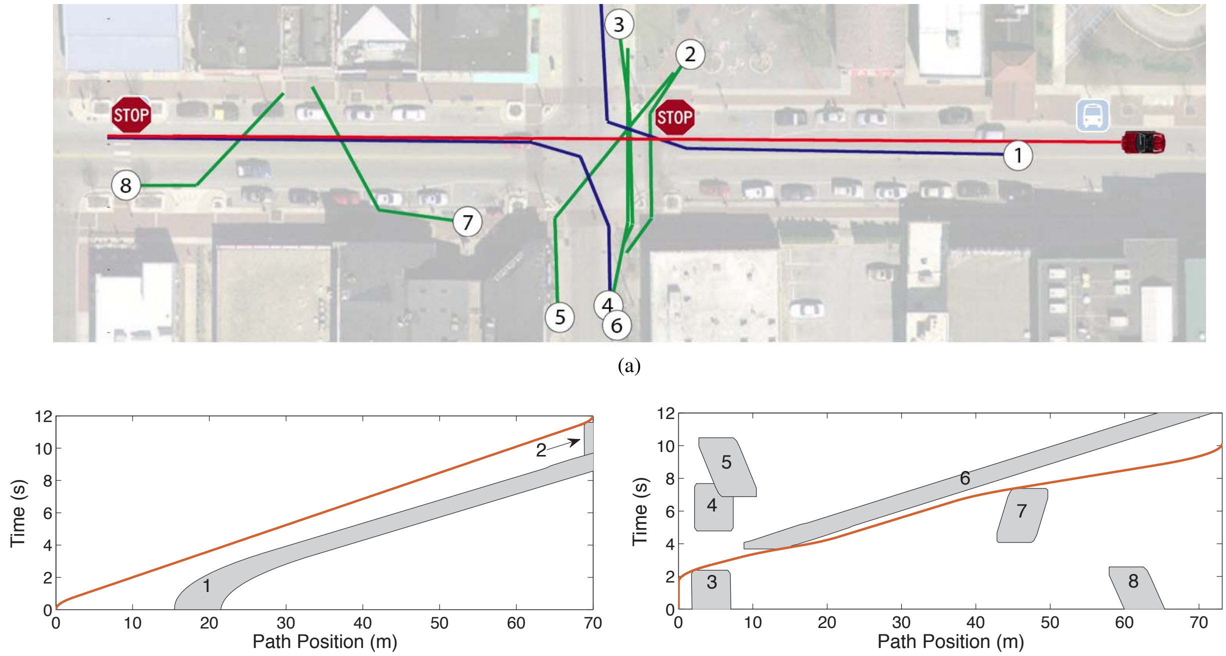
Fig. 5: Empirical performance

varying levels of discretization in PT obstacle construction, showing a roughly linear relationship. Fig. 5b shows how discretization affects the optimal goal arrival time. At coarse discretizations, narrow homotopy channels are occluded and the planner goes around them. At finer discretizations, these channels open up and the planner finds a faster route.

Fig. 5c compares our method to the A^*-based search method of [17]. Both planners were run on a simple scenario with zero to three obstacles. Velocity was constrained to be within $[0, 20]m/s$ and acceleration within $[-5, 5]m/s^2$, and time was discretized into steps of $0.1s$ for the search. The search heuristic is taken as the minimum remaining time to the goal position in the absence of obstacles. In the worst-case, the number of nodes A^* generates is exponential in the length of the trajectory, making it unsuitable for problems with long time horizons. On the scenario in Fig. 4 it fails to terminate after more than a minute.

V. APPLICATIONS

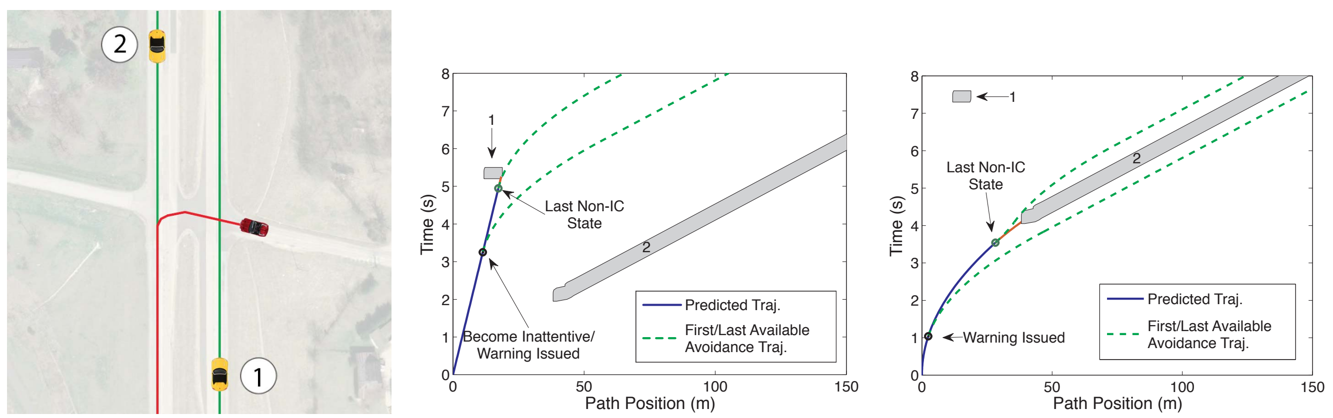
Autonomous Vehicles. We demonstrate the ability of our planner to handle the complex scenario of Fig. 4a[1]. We model a car traveling along on Kirkwood Avenue in Bloomington, Indiana on stretch of road with many restaurants and pubs.



(a)

(b) (c)

Fig. 4: (a) A complex urban driving scenario involving pedestrians and bicyclists. The car position, pedestrian and bicycle positions, and their paths are overlaid on the map. (b) PT obstacles of the stage 1 problem leading to the first stop sign, with the time-optimal trajectory shown in red. (c) PT obstacles of the stage 2 problem leading to the second stop sign.



(a) (b) (c)

Fig. 6: Illustration of collision warning. Fig. 6a: The driver (red) is merging onto the southbound lane of a rural highway, but fails to notice oncoming vehicles (numbered). Fig. 6b: With lead time $t_r = 2.5s$, an ICS is detected within t_r after the driver becomes inattentive and a collision warning is issued. Fig. 6c: The driver (red) accelerates too slowly while merging and a warning is issued at t_r from the first ICS.

Bicyclists often share the road lane and pedestrian traffic is heavy, both at and away from crosswalks. The problem is decomposed into two stages: 1) reaching the first stop sign, then 2) reaching a second stop sign. Acceleration bounds are $[-10, 8]m/s^2$ and velocity bounds are $[0, 13.4]m/s$. The supplemental video contains simulation of the scenario.

Fig. 4b shows the stage 1 trajectory. The car must avoid a bicycle (obstacle 1) moving in front of it. The bicycle accelerates from an initial stop, causing its PT obstacle to be curved initially, and then turns off the road after the stop, so its PT obstacle ends. Near the stop sign the car must avoid a pedestrian (obstacle 2) that cuts in front of the crosswalk.

Fig. 4c shows the stage 2 trajectory. The car must now avoid pedestrians in the crosswalk, as well as another bicycle (obstacle 6) that turns into the car's lane. The optimal plan has the car accelerate out in front of the bicycle, then

[1]Imagery ©2012 DigitalGlobe, GeoEye, IndianaMap Framework Data, USDA Farm Service Agency

decelerate slightly to avoid a pedestrian (obstacle 7) before going on to the final position.

Collision warning systems. Our planner can also be applied to collision warning systems for inattentive drivers. Suppose such a system can detect driver inattention and has a reaction time parameter t_r sufficient for a driver to perceive and respond to a warning, but not so long as to generate unnecessary false positives. Our planner can be called repeatedly to verify that a feasible trajectory exists, assuming the driver continues his/her current behavior up to time t_r. If not, then the driver is about to enter an *inevitable collision state* (ICS), and a warning is issued.

In order to do so, we first collision check the driver's predicted trajectory T_p up to time t_r. If a collision is found, a warning is issued. Otherwise, the planner attempts to find a feasible trajectory starting from the final state of T_p. If none is found, a warning is issued.

Consider a rural highway intersection scenario. The driver attempts to merge south onto State Road 37 in Indiana (Fig. 6a) after crossing two northbound lanes of traffic. The driver incorrectly judges the speed of a northbound vehicle and begins the merge too slowly to cross safely. The planner detects an ICS within t_r and a warning is issued to the driver at the point marked in Fig. 6b. With the appropriate indicators the driver would hopefully be able to accelerate out of the way of the vehicle, or an automated system might take over and guide the car to safety. In a second example with different initial conditions (Fig. 6c), the driver incorrectly judges the speed of the southbound vehicle and attempts to merge too slowly. The warning indicates that the driver should either slow down or stop at the median, or accelerate ahead of the oncoming vehicle. The supplemental video contains simulation of the scenario.

VI. CONCLUSION

We presented a complete, optimal longitudinal control planner in the presence of moving obstacles that extends previous work by allowing arbitrary polygonal models and agent trajectories. We demonstrated that it can plan time-optimal velocity profiles in cluttered scenarios and to detect inevitable collision states in collision warning systems. Simulation tests suggest that the planning system is fast enough for real-time navigation among many dynamic obstacles.

It is possible to further improve the speed of our planner, e.g., using efficient geometric data structures for accelerating PT obstacle construction or testing for collisions. We also hope to relax some of the assumptions behind our planner, such as allowing it to choose routes along a network of possible paths, and handling more realistic vehicle dynamic models and obstacle behavior models. Eventually, we intend to test our algorithms in more realistic driving simulators and/or real vehicles to better understand how they can be employed to improve driving safety.

The supplemental video for this paper is available at:
`http://www.indiana.edu/~motion/iv2013/`

REFERENCES

[1] "Roundabout: An informational guide," 2000, in U.S. Department of Transactions Federal Highway Administration Publication Number: FHWA-RD-00-067.

[2] T. C. Ng, J. Ibanez-Guzman, J. Shen, Z. Gong, H. Wang, and C. Cheng, "Vehicle following with obstacle avoidance capabilities in natural environments," in *Proceedings IEEE International Conference on Robotics and Automation*, vol. 5, april 2004, pp. 4283 – 4288 Vol.5.

[3] A. Uno, T. Sakaguchi, and S. Tsugawa, "A merging control algorithm based on inter-vehicle communication," in *Proceedings IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems*, 1999, pp. 783–787.

[4] J. Johnson and K. Hauser, "Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path," in *Proceedings IEEE International Conference on Robotics and Automation*, St. Paul, USA, May 2012.

[5] J. Hopcroft, D. Joseph, and S. Whitesides, "Movement problems for 2-dimensional linkages," *SIAM Journal Computing*, vol. 13, no. 3, pp. 610–629, Jul. 1984. [Online]. Available: http://dx.doi.org/10.1137/0213038

[6] P. Fiorini and Z. Shillert, "Motion planning in dynamic environments using velocity obstacles," *International Journal Robotics Research*, vol. 17, pp. 760–772, 1998.

[7] D. Wilkie, J. van den Berg, and D. Manocha, "Generalized velocity obstacles," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2009.

[8] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock, "Kinodynamic motion planning amidst moving obstacles," in *Proceedings IEEE International Conference on Robotics and Automation*, vol. 1, 2000, pp. 537–543.

[9] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[10] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, Oct. 2008.

[11] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[12] S. Kolski, *Mobile Robots: Perception & Navigation*. Pro Literatur Verlag, 2007.

[13] T. Besselmann and M. Morari, "Hybrid Parameter-Varying MPC for Autonomous Vehicle Steering," *European Journal of Control*, vol. 14, no. 5, pp. 418 – 431, 2008.

[14] P. Falcone, F. Borrelli, H. Tseng, J. Asgari, and D. Hrovat, "A hierarchical model predictive control framework for autonomous ground vehicles," in *American Control Conference*, june 2008, pp. 3719 – 3724.

[15] C. Urmson, J. Anhalt, J. A. D. Bagnell, C. R. Baker , R. E. Bittner, J. M. Dolan, D. Duggins, D. Ferguson, T. Galatali, H. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, A. Kelly, D. Kohanbash, M. Likhachev, N. Miller, K. Peterson, R. Rajkumar, P. Rybski, B. Salesky, S. Scherer, Y.-W. Seo, R. Simmons, S. Singh, J. M. Snider, A. T. Stentz, W. R. L. Whittaker, and J. Ziglar, "Tartan racing: A multi-modal approach to the darpa urban challenge," Robotics Institute, http://archive.darpa.mil/grandchallenge/, Tech. Rep. CMU-RI-TR-, April 2007.

[16] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: the path-velocity decomposition," *International Journal Robotics Research*, vol. 5, pp. 72–89, September 1986.

[17] T. Fraichard, "Dynamic trajectory planning with dynamic constraints: A 'state-time space' approach," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, jul 1993, pp. 1393–1400.

[18] C. Ó'Dúnlaing, "Motion planning with inertial constraints," *Algorithmica*, vol. 2, no. 1–4, pp. 431–475, 1987.